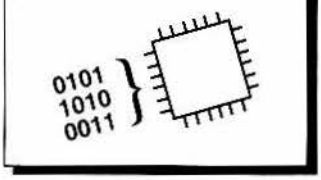


Parallel-port standards settle out in the '90s

Jim Kelsey

Cheap, simple and available in most PCs, the parallel port has lately become the focus of several industry initiatives. Perhaps most recognizable are the terms ECP and EPP, which enhance the port with hardware-driven bidirectional handshaking, background DMA and higher throughput. For many users, these schemes bring not only better performance but also increased frustration as they try to understand and correctly configure their systems to take advantage of these technologies. This column clarifies the current state of parallel-port initiatives and recommends how to get the most out of your port.

Two areas of parallel-port functionality need explanation. The first deals with modes the system I/O (super I/O) controller provides in hardware. The super I/O controller is a highly integrated ISA-bus device that contains the floppy controller, COM ports and LPT port and sometimes an IDE controller. Today, most systems use controllers that UMC, SMC, National Semiconductor or Acer Logic manufactures and supplies to motherboard manufacturers. The second area of functionality consists of configuration mechanisms the system BIOS and OS provide, meaning the user can get the hardware to execute in a particular mode.

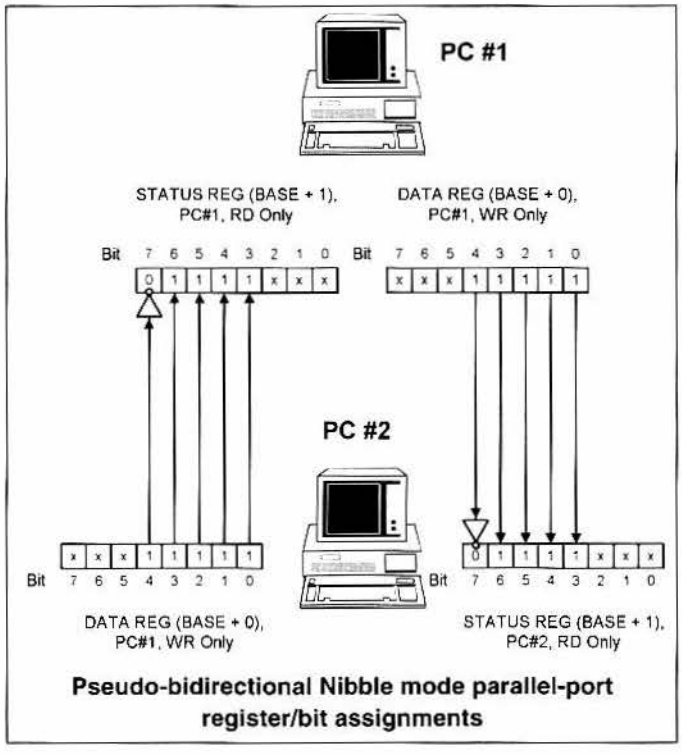
Most popular super I/O controllers support five parallel-port modes in hardware:

- SPP—Standard (IBM PC/AT) Parallel Port
- Fast Centronics—Centronics-defined, high-speed unidirectional transfers
- Bidirectional—PS/2-style 8-bit read/write data, also called Byte mode
- EPP—Enhanced Parallel Port
- ECP—Extended Capabilities Port

Often the BIOS-setup utility allows a user to configure the parallel port to operate in any of these modes. However, the BIOS can control the port mode only if the super I/O controller resides on the motherboard; the BIOS has no ability to detect and configure a super I/O controller resident on a plug-in adapter.

In addition, if the system BIOS and OS both support EPP or ECP, BIOS suppliers needn't provide options to select SPP, Fast Centronics or Byte modes during BIOS setup. The reason is because EPP and ECP modes each provide a control register that enables the OS to select the current parallel-port mode at run time. At a minimum, though, one BIOS-setup

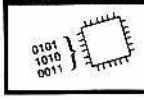
Fig 1—Nibble mode support relies on the large number of status lines available in the standard parallel-port pin out. Because these lines existed even in the first PCs, support for this mode is universally available.


PC #1
STATUS REG (BASE + 1),
PC#1, RD Only
DATA REG (BASE + 0),
PC#1, WR Only
Bit 7 6 5 4 3 2 1 0
0 1 1 1 1 x x x
x x x 1 1 1 1 1
PC #2
x x x 1 1 1 1 1
0 1 1 1 1 x x x
DATA REG (BASE + 0),
PC#1, WR Only
STATUS REG (BASE + 1),
PC#2, RD Only

Pseudo-bidirectional Nibble mode parallel-port register/bit assignments

Jim Kelsey is the director of platform development at SystemSoft Corp (Natick, MA (508) 647-2825). He's spent the last 10 yrs developing PC data-acq and system firmware products and is author of the book Programming Plug and Play.

EXPERT COLUMN
EMBEDDED



Signal Group	Signal Name	Direction	Signal Description
Control	nSTROBE	OUT	Active low. Indicates that the data on the data lines is valid.
	nAUTOFEED	OUT	Active low. Instructs the printer to automatically insert a line feed after each carriage return.
	nSELECTIN	OUT	Active low. Indicates to a printer that it has been selected.
	nINIT	OUT	Active low. Used to reset the printer.
Status	nACK	IN	The printer pulses this line low to signal successful receipt of a character.
	BUSY	IN	The printer asserts this line high to signal that it is busy and incapable of receiving data.
	PE	IN	The printer asserts this line high to signal that it is out of paper.
	SELECT	IN	The printer asserts this line high to indicate that it is online.
	nERROR	IN	The printer asserts this line when an error condition occurs.
Data	DATA[7:0]	OUT	Output only data lines.

Table 1—Standard Parallel Port signals

option allows users to enable SPP mode in case the OS has no built-in support for EPP or ECP.

Historical overview

Now let's take a closer look at each parallel-port scheme in more detail, starting with the early modes. Like its name suggests, SPP mode is the lowest common functionality a parallel port provides; in fact, it's the only mode in the original IBM PC/AT, so many people refer to it as "compatibility mode." The AT's designers intended SPP mode solely for sending data unidirectionally to a printer, plotter or similar device.

When operating in SPP mode, the parallel port transfers bytes of data to peripherals on lines $D_{0\text{-}7}$, and it receives status information from the peripheral on five input lines. System software accesses status data in SPP registers at offsets 0, 1 and 2 from the LPT-port base address. Table 1 shows the symbolic names for each SPP signal.

Unsatisfied with this scheme's unidirectional nature, many vendors in the late 1980s began to create peripherals capable of returning data to the host over the SPP's five status lines. This idea quickly grew into the concept of sending data bidirectionally between two computers using a special parallel cable. Inside such a cable, data bits DATA[4:0] on each end connect to the status inputs on the other end (Fig 1). A cable so wired allows applications to send or receive one nibble at a time using one status line as a handshake bit.

Not surprisingly, this parallel-port transfer method is called Nibble mode because drivers and applications can send data only one nibble at a time. Although slightly cumbersome, this mode is widely supported by such applications as Traveling Software's LapLink and Microsoft's INTERLNK.EXE driver.

To the dismay of today's OS designers, Nibble mode handshaking and data transfers consume considerable CPU bandwidth, thereby inhibiting the execution of other OS threads. In addition to hogging CPU time, bidirectional Nibble mode achieves a maximum transfer rate of <100k bytes/sec.

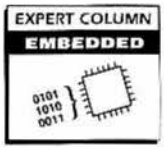
The first higher-speed effort

To address these performance limitations, Intel, Xircom and Zenith Data Systems introduced EPP mode to provide high-speed, bidirectional host-peripheral data transfers with peripheral-generated interrupt capabilities. Intel first introduced the technology in the 82360 controller in 1992. As Table 2 shows, EPP implementation redefines the function of SPP pins.

To use an EPP-capable device to its fullest potential, a user must enable EPP mode within the super I/O controller. Most EPP-capable systems provide a BIOS-setup

SPP Signal	EPP Signal	Direction	Signal Description
nSTROBE	nWRITE	OUT	Low indicates write operation, high indicates read.
nAUTOFEED	nDATASTB	OUT	Active low. Indicates a data R/W cycle is in progress.
nSELECTIN	nADDRSTB	OUT	Active low. Indicates an address R/W cycle is in progress.
nINIT	nRESET	OUT	When pulsed low, the peripheral resets.
nACK	nINTR	IN	When pulsed, the peripheral is requesting interrupt service from the host.
BUSY	nWAIT	IN	Handshake signal. When low, nWAIT signals that it is OK for the host to start a data or address cycle (assert a strobe). When high, nWAIT signals that it is OK for the host to complete a data or address cycle (de-assert a strobe).
DATA[7:0]	AD[7:0]	IN/OUT	Bidirectional address/data lines.
PE	Vendor specific	IN	Specific to each EPP-capable peripheral.
SELECT	Vendor specific	IN	Specific to each EPP-capable peripheral.
nERROR	Vendor specific	IN	Specific to each EPP-capable peripheral.

Table 2—Comparison of Standard and Enhanced Parallel Port signals



Calling Mechanism	Function Name	Description
INT 17H	Installation Check	Returns EPP version and EPP Vector (API entry point).
EPP Vector	QueryConfig	Return the capabilities and hardware version of the EPP.
EPP Vector	SetMode	Set the operating mode of the EPP to SPP, EPP, ECP, etc.
EPP Vector	GetMode	Get the current operating mode of the EPP.
EPP Vector	InterruptControl	Enable/disable EPP-interrupt generation.
EPP Vector	EPPReset	Reset the peripheral connected to the EPP.
EPP Vector	AddrWrite	Perform an address-write/device-select I/O cycle.
EPP Vector	AddrRead	Perform an address-read I/O cycle.
EPP Vector	WriteByte	Write a single byte via the EPP.
EPP Vector	WriteBlock	Write a block of data via the EPP.
EPP Vector	ReadByte	Read a single byte of data from the EPP.
EPP Vector	ReadBlock	Read a block of data from the EPP.
EPP Vector	AddrReadByte	Combination of AddrWrite and ReadByte functions.
EPP Vector	AddrWriteByte	Combination of AddrWrite and WriteByte functions.
EPP Vector	AddrBlockRead	Combination of AddrWrite and ReadBlock functions.
EPP Vector	AddrBlockWrite	Combination of AddrWrite and WriteBlock functions.
EPP Vector	LockPort	Disallow EPP multiplexer functions to occur.
EPP Vector	UnlockPort	Allow EPP multiplexer functions to occur.
EPP Vector	EPPInterrupt	Install an interrupt vector for an EPP-aware device driver.
EPP Vector	RealTimeMode	Query if time-critical devices are attached to EPP.

Table 3—EPP BIOS functions

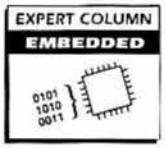
option for doing so directly. Users, however, can also enable EPP support indirectly by loading device drivers that either write to the controller chip's configuration registers or invoke BIOS functions to configure and initialize that mode.

Also note that EPP is backward compatible with SPP at the register level. If a user has configured the port for EPP, mode-specific registers appear at offsets 3-7 from the LPT-port base address, so enabling this mode doesn't affect the functioning of SPP registers at offsets 0-2. Reading and writing data through the EPP registers generate EPP-specific, rather than SPP-style, handshaking on the port pins. The OS driver must then determine the capabilities of the attached peripheral and access EPP or SPP registers accordingly.

Despite its backward compatibility, better throughput and bidirectional capabilities, hardware and software support for EPP was scarce. A year after that scheme appeared, a loosely structured industry group called the EPP Committee formed to define the EPP BIOS. This interface would automate the configuration and use of EPP-capable systems and peripherals. The system firmware provides the EPP BIOS interface except for multiplexer functions, which reside in a vendor-specific device driver. This firmware also includes a Real mode far-call interface for configuring and transferring data to and from EPP devices.

More importantly, an EPP BIOS is designed to transfer data using the fastest available mode. In the worst case, a port implements only SPP signaling. In that case, the EPP BIOS Committee assumed BIOS vendors would build emulation modes into their firmware by sending and receiving data in Nibble mode, although few vendors did so. With this emulation, in theory, OS drivers can perform all transfers through the EPP BIOS, and peripheral vendors need only supply an EPP-capable driver. This on-the-fly software adaptation never gained market acceptance because peripheral vendors still must provide drivers for systems that lack an EPP BIOS.

Applications that do employ this mode use the EPP BIOS interface by first issuing INT 17H, AH = 02H, the EPP Installation Check service. Upon successful completion, that service returns the EPP Vector in registers ES:BX. Applications request all sub-



SPP Signal	ECP Signal	Direction	Description
nSTROBE	HostClk	OUT	Together with PeriphAck, HostClk is used to transfer address or data informaton in the forward direction.
nAUTOFEED	HostAck	OUT	In the forward direction, provides Command/Data status. Together with PeriphClk, HostAckOut is used to transfer data in the reverse direction.
nSELECTIN	1284Active	OUT	If high, the host is operating in a 1284 transfer mode.
nINIT	nReverseReq	OUT	If low, the channel switches to reverse direction.
BUSY	PeriphAck	IN	Together with HostClk, PeriphAck is used to transfer address or data information in the forward direction. In the reverse direction, PeriphAck provides Command/Data status information.
PE	nAckReverse	IN	The peripheral drives this signal low to acknowledge a change to reverse direction.
SELECT	Xflag	IN	Extensibilty flag used to query peripheral capabilities.
nERROR	nPeriphRequest	IN	The peripheral drives this signal low to indicate that reverse data is available.
DATA[7:0]	DATA[7:0]	IN/OUT	Used to transfer data between host and peripheral.

Table 4—Comparison of ECP signals to their SPP counterparts

sequent EPP BIOS services by issuing far calls to the location that vector specifies. The authors of the EPP BIOS Specification never enhanced that document to support a runtime Protected mode interface, so its support is limited to Real or Virtual X86 mode applications.

The EPP architecture handles multidevice addressing through an external multiplexer or daisychaining logic. The design of such external devices is vendor specific. And although the EPP BIOS spec defines a set of extensions to support multiplexing/daisychaining peripherals, a ROM-based EPP BIOS doesn't provide them. Instead, individual vendors must offer drivers that support multiport extensions appropriately for their hardware.

Because EPP allows multiple devices on one parallel port, it's important that the BIOS implement a locking capability to prevent several drivers from simultaneously using the BIOS. To do so, the BIOS provides both a LockPort and UnlockPort function. A complete list of EPP BIOS functions, excluding multiport extensions appears in Table 3.

Despite its promising potential, EPP has met some daunting roadblocks. For example, the EPP BIOS's real-mode only API prohibits its use under Windows NT. Also, Microsoft's refusal to back EPP has inhibited the technology's exposure to vendors and end users alike.

The former EPP BIOS committee chairman Larry Stein admits that the original EPP BIOS initiative was overly ambitious, but maintains that EPP continues to make inroads in the non-printing peripheral-market segment. According to Stein, vendors such as Hewlett-Packard and Iomega are producing drivers that intelligently circumvent the need for an EPP BIOS by directly configuring the PC's parallel port(s). He also hinted that industry adoption of the proposed IEEE 1284.3 specification, which contains a scaled-down EPP BIOS, would go far to alleviate current EPP BIOS woes.

The latest and greatest

The newest parallel-port initiative, ECP enjoys the focus and support of industry leaders Hewlett-Packard and Microsoft. Unlike EPP, it offers a standardized register interface as well as features not available in other schemes such as hardware data compression and DMA. Additionally, while EPP requires peripheral device drivers to handle port sharing (say between a printer and a parallel-port SCSI adapter), ECP doesn't place this burden on device driver authors. And, unlike the others, ECP has flourished. Originally an IEEE initiative (P1284), Microsoft and HP refined and defined the scheme in areas the IEEE spec was unable to address. Among them are such features as RLE (run-length encoding) compression, forward/reverse-channel FIFOs and

Mode	Mode Description
000b	Standard Parallel Port (SPP) Mode
001b	PS/2 bidirectional (BYTE) Mode
010b	Fast Centronics Mode
011b	Extended Capabilities Port (ECP) Mode
100b	Enhanced Parallel Port (EPP) Mode
101b	Reserved/Unused
110b	ECP Test Mode
111b	ECP Configuration Mode

Table 5—Extended Capabilities Port modes

Offset	Register Name/Function	R/W	Modes in which register is valid
000H	Data	R/W	000b, 001b
000H	ECPAFifo (Address FIFO)	R/W	011b
001H	Status	R/W	All
002H	Control	R/W	All
400H	DataFIFO	R/W	011b
400H	TestFIFO	R/W	110b
400H	ConfigRegA	R/W	111b
401H	ConfigRegB	R/W	111b
402H	ExtControl	R/W	All

Table 6—ECP registers

DMA. Microsoft and HP published their interpretation of the spec along with recommendations for adapting it for PC-industry use in a document entitled "Extended Capabilities Port: Specification Kit," available from Microsoft (Redmond, WA (425) 882-8080).

Like SPP and EPP, ECP includes its own definition for physical signals (Table 4). As does EPP, this newer scheme implements hardware handshaking between a host and a peripheral. Operationally, though, it differs from EPP in several respects. First, ECP data transfers are preceded by a direction negotiation phase. The host and peripheral negotiate the direction of the next operation with the nPeriphRequest and nPeriphAck signals.

Next, once the host and peripheral have established the transfer direction, each uses PerhiphAck, HostClk, PeriphClk and HostAckOut to handshake forward/reverse-direction transactions. Because of the overhead required to change transfer direction, ECP is most efficient when sending or receiving large blocks of data. System software then transfers data with either programmed I/O or DMA. The ECP FIFO register resides at offset 400H from the LPT-port base address. This placement prevents register contention between ECP and traditional ISA devices, which are capable of only 10-bit I/O address decoding. Tables 5 and 6 show ECP modes and the register definitions for each. Note that system software sets the current parallel-port mode in the register at offset 402H.

Not everyone, though, sets their systems up to handle ECP the same. Users should configure the BIOS setup utility with its parallel-port default mode as ECP only if the OS comes with a driver that can switch among the modes in Table 3. Win95, for instance, includes the LPT.VxD driver for this purpose. It sets the port operating mode based on the capabilities of the attached peripheral. To support LPT.VxD, the system Plug and Play BIOS must contain a device node whose device identifier indicates that the parallel port is ECP capable. If the user selects ECP mode during setup, LPT.VxD launches in ECP mode and later switches to SPP as required. Note that LPT.VxD doesn't support EPP.

Additionally, the Plug-and-Play BIOS provides different IRQ and DMA combinations to allow Win95 and other Plug-and-Play operating systems to allocate/deallocate resources as required. An ECP operating in DMA mode requires at minimum that the Plug-and-Play BIOS offer two DMA channel selections and two IRQ selections. **PE&IN**

References

1. EPP BIOS Specification, Rev. 3.0, http://www.fapo.com.
2. IEEE P1284 Specification, "Standard Signaling Method for a Bi-directional Parallel Peripheral Interface for Personal Computers" IEEE Standards Department, Piscataway, NJ.

Editorial Feedback
This article's value to me was:
High—269 Average—270 Low—271